Manuel J. A. Eugster

Programming R

Chapter 1: Basic Vocabulary

Last modification on April 11, 2012

Draft of the R programming book I always wanted to read
http://mjaeugster.github.com/progr

1 Basic Vocabulary

The basic vocabulary you should know to read this book.

The main literature for this section is:

- *Software for Data Analysis: Programming with R* by Chambers (2008)
- *An Introduction to R* by R Core Team (2012a)
- *R Language Definition* by R Core Team (2012b)

1.1 Session

Console: R is a command line interpreter—when you start R, it issues a prompt waiting for input. The default prompt is "`> `".

```
> 1 + 1


[1] 2
```

The input is called an **expression**. If the user gives a complete expression, R evaluates the expression and shows the output immediately. If—after the user has pressed `Enter`—the input can not be interpreted as a complete expression, R keeps prompting for more input showing a "`+ `".

```
> 1 +
+ 2


[1] 3
```

If the output is a vector R indicates the index of the left-most element in each line; in the example above with `[1]`.

See also: `?options` to change global options which affect the way in which R computes and displays its results.

Session: A session starts when you start up R and ends when you quit R.

```
> q()
```

The function to quit the current R session asks if you want to save the **workspace** of the current session.

Objects: During an R session objects are created and stored by name (case-sensitive).

```
> a <- 1 + 1
> a

[1] 2

> b <- "Hello World!"
> b

[1] "Hello World!"
```

The function `ls()` (or `objects()`) lists all "currently stored" objects.

```
> ls()

[1] "a"   "b"   "txt"
```

To remove one object use the `rm()` function.

```
> rm(a)
> ls()

[1] "b"   "txt"
```

Execute `rm(list = ls())` to remove all currently stored objects.

Workspace: The **environment** of an R session is called the workspace. There, all objects of a session are stored. Use `ls()` to list all objects of the current workspace.

At the end of an R session (indicated, e.g., by executing `q()`), the user is asked to save the current workspace or not. If approved, the objects are written to a file called `.RData` in the **current working directory**, and the command lines used in the session are saved to a file called `.Rhistory`.

When R is started at later time from the same directory it automatically reloads the workspace from this file and the associated commands history is reloaded.

Note that files starting with a `.` are normally hidden files, and you have to explicitly indicate in your operating system (or shell, or Explorer, etc.) that you want to see these files.

See also: Use the function `save.image()` to save the current workspace at any time during a session. Use the `save()` function to save specific objects.

Working directory: The working directory is the default directory where all "things" (workspace, figures, etc.) are stored or looked for (source files, etc.) when no specific path is stated.

The function `getwd()` returns the current workspace, the function `setwd()` sets the current workspace.

```
> setwd("z:/Projects/progr/progr")

Error: cannot change working directory

> getwd()

[1] "/Users/manuel/Projects/progr/progr"
```

It is useful to create a working directory for each project.

Note that in Microsoft Windows you have to use a slash (`/`) or a quoted backslash (\\) instead of a backslash (\).

1.2 Help

`?`: For each user-visible object (function, data set, etc.; see the chapter about namespaces to learn what user-visible means) a help page with detailed information is available. To get the information for a specific object, use the function `help()` or the shortcut `?`.

```
> help(ls)
> ?ls
```

Note that keywords and special functions (e.g., `if` or `[[`) must be enclosed in quotes.

```
> ?"if"
```

Help topic: A help page always contains a description, a usage section and a description of all arguments. It can contain a details section, references and links to similar functions (the "see also" section).

Often an example section demonstrates the function's usage; the examples on a help topic can normally be run by `example("topic")`.

```
> example("ls")
```

Search: If the the concrete function name, data set name, etc. is unknown, one can search the complete help system using the function `help.search` or the shortcut `??`.

```
> help.search("linear models")
> ??"subset"
```

This results in a list with matched objects, each entry of the type `PKG::FOO`. `FOO` is the the name of the function, data set, etc. and `PKG` the package where `FOO` is in.

Note that the option `help.search.types` specifies the different "things" to search through.

```
> getOption("help.search.types")

[1] "vignette" "demo"     "help"
```

See also: The add-on package `sos` provides extended search functionality to quickly and flexibly search and find help pages, data sets, variables in data sets, etc.

Hypertext Documentation: R is shipped with a comprehensive hypertext documentation; execute `help.start()` to open a browser with the HTML version of R's online documentation. There you find Manuals, the documentation for all packages, etc.

1.3 Language

Object: Everything in R is an object; and these objects are referred to through **symbols** (often also called names or variables). The symbols are case-sensitive.

```
> x <- 1
> x


[1] 1


> X <- "a"
> X


[1] "a"
```

In this example `x` is the symbol which refers to an object with the value `1` and `X` is the symbol which refers to an object with the values `a`.

Function: Functions (and similar constructs) are the workhorses of R. Functions are invoked by name followed by a list of **arguments** separated by commas in parentheses. Each function has a **return value**, i.e., the result of the computations in the function as an R object.

```
> identity(1:3)


[1] 1 2 3
```

In this example the function `identity()` is called with one argument, the vector of integers from 1 to 3. As this function is the identity, the return value is the same vector.

Note that if the function name given without parentheses, the source code of the function is displayed.

```
> identity


function (x)
x
<bytecode: 0x13166bc>
<environment: namespace:base>
```

Operator: Operators are functions with a different calling syntax. Typical operators are the binary arithmetic operators `+`, `-`, etc. or the unary arithmetic operator `-`.

```
> 1 + 1

[1] 2

> -1

[1] -1
```

Operators also can be called like ordinary functions by quoting the operator names using backticks `‘op‘`.

```
> ‘+‘(1, 1)

[1] 2

> ‘-‘(1)

[1] -1
```

See also: `?Syntax` for operator syntax and precedence; follow the “See Also” to arithmetic, relational, logical, etc. operators.

Vector: The most basic objects are vectors of various types, e.g., numbers. There are no scalar object types underlying them. Single numbers (or other types) are still vectors, of length 1.

```
> x <- 4.2
> is.vector(x)

[1] TRUE

> length(x)

[1] 1
```

This concept allows **vectorizing computations**—the execution of an expression for each element of vector without using an explicit loop.

```
> x <- c(4.2, 13, 5)
> cos(x)

[1] -0.4903  0.9074  0.2837

> x + 1

[1]  5.2 14.0  6.0
```

Comment: Comments are ignored by R. Any text from a # character to the end of the line is taken to be a comment.

```
> # This is a comment line
> 1 + 1  # A Computation

[1] 2
```

1.4 Packages

Package: All R functions and datasets are stored in packages. Only when a package is loaded (attached) its content is available.

Execute `search()` to see which packages are currently loaded and therefore which functions, data sets, etc. are available to use.

```
> search()

 [1] ".GlobalEnv"        "package:knitr"     "package:stats"
 [4] "package:graphics"  "package:grDevices" "package:utils"
 [7] "package:datasets"  "package:methods"   "Autoloads"
[10] "package:base"
```

In order to see which packages are installed and available to load, issue the function `library()` function without an argument. To load a package use the `library()` or `require()` functions and the package's name as first argument.

```
> library("tools")
```

Execute `detach("package:tools")` to unload a package.

See also: `.packages()` returns information about package availability.

Package installation: The R distribution comes with some base packages. Other add-on packages have to be installed from a **repository** like CRAN (the default repository).

```
> install.packages("archetypes")
```

In this example a package with the name `archetypes` is installed from CRAN into a local **library tree**.

See also: Use `update.packages()` to update all packages; or `download.packages()` to just download a package without installation.

Repository: Repositories are places (websites) where a set of packages are hosted and ready for installation. Prominent ones are CRAN and Bioconductor.

A global option defines the default repository used by `install.packages()`.

```
> getOption("repos")

    CRAN
"@CRAN@"
```

See also: `chooseCRANmirror()` to set a CRAN mirror.

Library: Libraries or library trees are local directories where packages are installed (by `install.packages()`) and looked for (by `library()` and `require()`). Execute `.libPaths()` to see the current paths.

```
> .libPaths()

[1] "/Users/manuel/Library/R/2.14/library"
[2] "/Library/Frameworks/R.framework/Versions/2.14/Resources/library"
```

The list of paths is ordered, i.e., R tries to install a package in the first directory, then in the second, etc.

See also: See `?.libPaths` for the environment variables which define these paths.

Package content: An R package typically contains some functions, data sets, and documentation of both. To get an overview execute `library(help = "package")` or look at its hypertext documentation (execute `help.start()` and follow the links "Packages" and the package title).

```
> library(help = "archetypes")
```

To get a list of a package's data sets and to load a specific one use the `data()` function.

```
> data(package = "archetypes")
> data("body", package = "archetypes")
```

Demos are longer examples that demonstrate some of the functionality of the package. To get a list of a package's demos and to execute a specific one use the `demo()` function.

```
> demo(package = "archetypes")
> demo("robust-ozone", package = "archetypes")
```

To see the demo's source code execute:

```
> edit(file = system.file("demo", "robust-ozone.R",
+      package = "archetypes"))
```

Vignettes are (typically) detailed descriptions of parts of a package. To get a list of a package's vignettes and to open a specific one use the `vignette()` function.

```
> vignette(package = "archetypes")
> vignette("archetypes", package = "archetypes")
```

To see the vignette's source code execute:

```
> edit(vignette("archetypes", package = "archetypes"))
```

See also: `browseVignettes()` lists available vignettes in an HTML browser with links to PDF, LaTeX/noweb source, and (tangled) R code (if available).

1.5 Scripts

R, as already said, uses a simple command line interface. In fact, one operates in a (infinite) **read-eval-print loop** (REPL): the user enters an expression (read), the expression is evaluated (eval), and the result is displayed (print)—and this is repeated until the user enters `q()`.

This interactivity is nice to learn the language, to explore R and to do short analysis, however, it is unhandy to make long, complex, and reproducible analysis or to enhance R with own functionality. Therefore, it is much more comfortable to maintain all expressions in one or more separate **script** files (with the ending `.R`).

Let's say a script file `basic-1.R` is in the current working directory with the following content:

```
a <- 2:8
b <- pi
c <- a * b
save(c, file = "res.RData")
```

Then one can execute the complete script at once by using the `source()` function.

```
> source("basic-1.R")
```

The usage of R scripts has the advantage that all steps of a project are collected in one (or more) script files, changes are very easy to accomplish, and each step of the project can be reproduced at any time. The loss of interactivity can be regained by using an appropriate **editor** for R scripts.

1.6 Editors

Programmers need the best tools to do their job. One important tool is the editor they use to write source code. Appropriate editors for R scripts (in my point of view) are **RStudio** (RStudio, Inc, 2012) and **Emacs+ESS** (Rossini et al., 2012).

Bibliography

John Chambers. *Software for Data Analysis: Programming with R*. Springer, 2008. ISBN 9780387759357.

R Core Team. *An Introduction to R*, 2012a. URL http://cran.r-project.org/doc/manuals/R-intro.html.

R Core Team. *The R language definition*, 2012b. URL http://cran.r-project.org/doc/manuals/R-lang.html.

Anthony J. Rossini, Martin Maechler, Rodney A. Sparapani, Stephen Eglen, Richard M. Heiberger, and Kurt Hornik. Emacs Speaks Statistics, 2012. URL http://ess.r-project.org/.

RStudio, Inc. RStudio, 2012. URL http://rstudio.org/.